IOWA STATE UNIVERSITY
Digital Repository

Graduate Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2019

Using a system dynamics simulation model to explore the effect of information delays on software development

Xinxin Yang
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation

Yang, Xinxin, "Using a system dynamics simulation model to explore the effect of information delays on software development" (2019). *Graduate Theses and Dissertations*. 17619.
https://lib.dr.iastate.edu/etd/17619

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Using a system dynamics simulation model to explore the effect of information delays on software development

by

Xinxin Yang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Carl K. Chang, Major Professor
Simanta Mitra
Ali Jannesari

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

Copyright © Xinxin Yang, 2019. All rights reserved.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my advisor, Professor Carl K. Chang, and my committee members, Dr. Simanta Mitra and Dr. Ali Jannesari, for their guidance, enthusiastic encouragement and useful critiques throughout the course of this research.

I would also like to thank Mr. Robert Ward for his valuable and constructive suggestions during the planning and development of this research work. His willingness to give his time so generously has been very much appreciated.

In addition, I would also like to thank my friends, the department faculty and staff for making my time at Iowa State University a wonderful experience.

ABSTRACT

Software development decisions are made based on the information collected during the development process. Information delay is one of the major reasons that developers make wrong decisions. Slack time is important for development team to be creative and productive. Shortened information delay and properly allocated slack time can improve project performance. System dynamics modeling can provide insights at both macro and micro level. It includes quantitative representation, feedback and control system.

The development of the model was guided by hypothesis that employees make inappropriate decisions about work intensity because they receive inaccurate information about the project status. The effects of independent and dependent variables on the model have been analyzed separately.

The primary contribution of this research is to propose a strategy for organizations to handle the information delay and accelerate the availability of accurate information as simple as changing the behavior of the project. There is zero cost change through the process. All the changes can apply to a traditional model happen to be exactly the same change in Scrum.

CHAPTER 1. INTRODUCTION

Many software development organizations struggle with project productivity, completion delay, terminal deadline, and customer dissatisfaction. These troubles, to some extent, come from the wrong decisions made by both management and employees during the development process. One reason that employees make wrong decisions is due to the inaccurate information they have received about the project status. In particular, delayed information about project progress tends to lead to inappropriate decisions about work intensity. Human nature dictates that the increased productivity effects of overtime can only be temporary. People can work a little harder under pressure, but after a while fatigue sets in and productivity decreases dramatically.

System Dynamics simulation was created to enable quantitative representation and simulation of feedback and control systems [1][2]. Almost all human endeavors, including software development, involve significant feedback paths. Dynamic feedback loops are very valuable in modeling in terms of capturing the interaction of different variables [3][4]. A dynamic hypothesis is a working theory of how a problem will be represented by simulation model-based relationships evident in the real world. System dynamics addresses process improvement and management particularly well. It also helps facilitate human understanding and communication of the process but does not explicitly consider automated process guidance or automated execution support [1][5]. It is a hypothesis because it is provisional, subject to revision or abandonment as we learn from the modeling process and from the real world [2]. The first step in the development of an initial system domain model is the identification of the dynamic hypothesis that defines the scope of the problem being modeled. The hypothesis on identifying structural factors: Employees make inappropriate

decisions about work intensity because they receive inaccurate information about the project status.

My research is to propose structural and policy adjustments that can improve behavior through identifying information delays, especially in feedback loops related to work intensity which also affect slack time allocation, work overtime, and projected schedule profiles. This paper will describe this research in five sections:

1. Problem identification and definition
2. Approach of conducting experiments
3. Model formulation
4. Model evaluation
5. Research limitation

CHAPTER 2. EFFECTS OF INFORMATION DELAY IN SLACK TIME USE

Problem Indemnification and Definition

Slack time, as the opposite of overtime, is necessary for people to get inspired for new ideas and be creative [1]. Slack time is the primary resource for responding to small surprises, for dealing with unavoidable randomness. Wise allocation of slack time is important in the software development process. We want to keep the slack time in a certain range, so employees can respond to pressure by trimming waste and focusing on the critical path instead of feeling unbearable pressure and starting to look for other work. When employees receive inaccurate information about project status, they may allocate slack time inappropriately. However, the wrong information is not simply released by people from management team, but also is a consequence of the way the process is structured. My research is to identify potential problems in project structures.

Approach of Conducting Experiments

Feedback systems can capture the inner relations between causes and effects. A Feedback loop is a closed sequence of causes and effects, a closed path of actions and information. An interconnected set of feedback loops is a feedback system. System dynamics modeling can provide insights at a macro or micro level of software development process [1]-[4]. System dynamics can be widely used to virtually investigate different life-cycle processes, defect detection techniques, decisions on testing, criteria and so on.

Simulation has been widely used to support interpretation of software development. It provides a feasible experimental tool for testing software engineering. Simulation is less expensive, faster, and easier to control than field research. Simulation can help reduce the risk of software development. Organizations can concentrate on specific aspects of development planning, costs, product quality, schedule, depending on their concerns. Abdel-

Hamid was the first researcher to apply system dynamic methodology to a comprehensive model of software project management. Abdel-Hamid and his researchers have studied a variety of issues in the software development process based on a system dynamics simulation model in the context of plan-based traditional software development [7]. I believe that system dynamics simulation modeling can help us gain valuable insights about project productivity lose and completion delay.

System dynamics modeling gives us the ability to conduct multiple simulations and compare the performance of each simulation. One of the advantages of system dynamics is that it is easy to change the value of one parameter without affecting other parameters. In practice, the control of a particular variable is hard to obtain. In this research, we can simply modify one variable in one simulation, and compare the effects on the project performance. We are trying to propose an approach to improve the overall performance through the comparison of differently parameterized runs of the same simulation model.

Abdel-Hamid's model has fully represented the entire software development life cycle except for design. It uses the feedback principles of system dynamics to structure and clarify the complex web of dynamically interacting variables [8]. Modeling the development processes will help answer difficult questions that researchers and managers face in balancing the different needs for quality and speed and help tailor processes as the organizational and development environments change [3].

Model Formulation

Abdel-Hamid's model is an integrative system dynamics model. As shown in Figure *1*, his model integrates not only management type functions (e.g., planning, controlling, and staffing) but also production type functions (e.g., coding, reviewing, and testing). Abdel-Hamid's model covers four different aspects of software development projects: human

resource management, software production, planning and controlling. There are four main activities in software production subsystem in Abdel-Hamid's model: development, quality assurance, rework, and system testing. Abdel-Hamid's goal is to analyze the effects of project management policies on software development. His model has been widely used to investigate software cost and schedule estimation, economics of the quality assurance function, project staffing, and other issues such as software reuse, project control with fallible information [2].

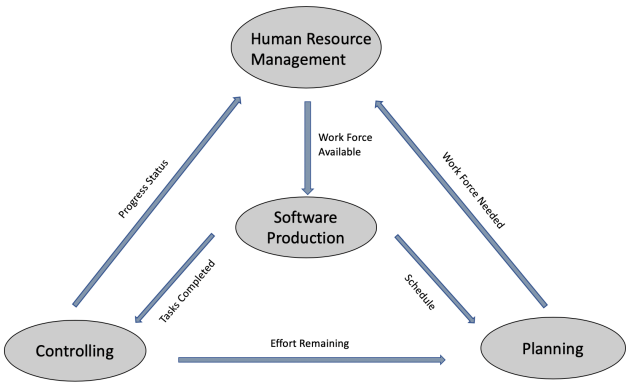


Figure 1: *Abdel-Hamid Software Development Subsystems*

The original Abdel-Hamid model was written in Dynamo, which is no longer supported. I used a model in Vensim created by semi-automatic translation from the original and validated against the published results from the original dynamo [9]. My simulations reuse Abdel-Hamid's model with different values for parameters that represent certain information delays: specifically delays that might affect employee decisions about work intensity and slack time. Software development organizations underestimate the project in most cases. Even in Abdel-Hamid's model, the project was initially estimated completion time as 360 man-days, it actually took around 420 man-days to be completed. Abdel-Hamid's model accurately models the real world experience of projects and project managers: as shown in Figure *2*, initially the project seems can be completed a little early,

but as the project progresses, more undiscovered tasks discovered per day which leads to an overshoot of the man days remaining, and gradually dropping until the project completed.

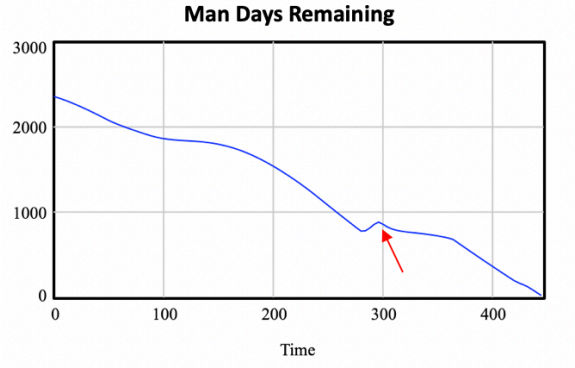


Figure 2: *Project Man-Days Remaining*

Model Evaluation

"Slack time is the fraction of project time lost on non-project activities, e.g., coffee-breaks, personal business." [8] Slack time is necessary in software development. It gives the development team flexibility and elasticity to handle unexpected 'surprises' during the development process. It allows the team to quickly respond to unplanned work without requiring overtime efforts and risking employee burnout [10].

Our goal is to understand how various delays affect the organization's adaptability. We are especially interested in the effect of these delays on workers' slack time and on the need for overtime. Once employees have invested their time in slack activities, that discretionary time is no longer available to fund further adaptive behavior. The later employees receive information which indicates they need to work harder, the less likely they will be able to work harder, for long enough, to get the project on track.

Figure *3* represents the work intensity variables in the original Abdel-Hamid's model. At day 282 management becomes aware that the original estimate is too low and at Day 274 system integration testing begins and Days intervals 280-296 and 420-434 represent periods

of extreme work-intensity as employees try to get “back on plan”. The interval between the two periods of intense work reflects a recovery period following a burnout cycle.

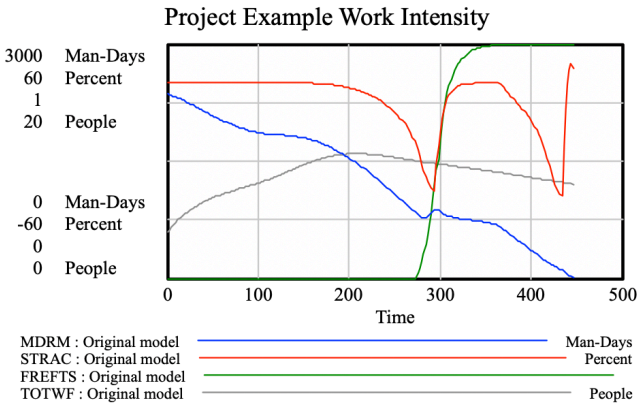


Figure 3*: Abdel-Hamid Model Project Work Intensity*

To support this line of inquiry, I examined delay variables in Abdel-Hamid’s model and selected those which seemed likely to influence work-intensity decisions. I analyzed the impact of each of these delays by modifying each value, simulating the process and graphing the resulting behavior. Two types of variables have been applied in Abdel-Hamid’s model: independent variables and dependent variables. The unifying characteristic in the variables (independent variables) I am adjusting is that they affect delays in information flow. I chose to monitor, as dependent variables, slack time, overtime, and traditional project cost, time and quality factors.

Table 1 shows all the independent variables modified in my experiments. Compared to the original values used in the Abdel-Hamid’s model, greater and smaller values have been applied to these variables to evaluate the model performance.

Table 1: *Parameter Values applied in the Model Simulations*

Subsystem			A-H Model	Simulation A	Simulation B
Software Production Subsystem	Manpower Allocation Sector	Desired Rework Delay (DESRWD)	15	25	5
	Quality Assurance and Rework Sector	Average Delay for QA (AQADLY)	10	15	5
Control Subsystem	Reporting Delay (RPTDLY)		10	20	5
	Average Delay in Incorporating Discovered Tasks (DLINCT)		10	15	5

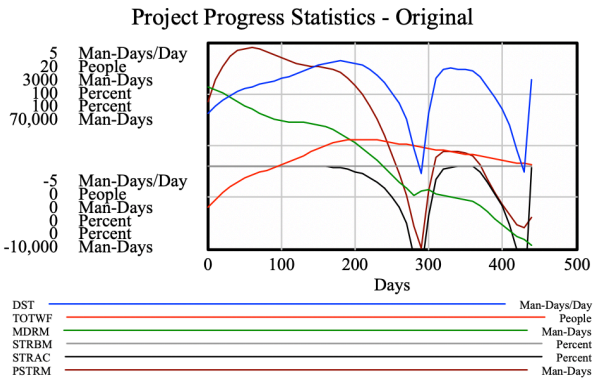
Meantime, I examined the Fraction of Effort for System Testing (as FREFTS in the equations) in the model. FREFTS is the output of a function which specifies the allocation of manpower between development and testing, implicitly, the timing of starting system integration testing. I want to know the correlation between information delay and system integration testing starting time, then identify the best time to start system integration testing, since test results form an important source of information about project status.

I want to gain some insights by observing the changes of tendency correlated to the modification of each variable. As you may notice I compared the outcome with large values, small value and the values used in the original model. It may not be a practical software development approach in the real world, but with rapid changes in values, we can have a better visualization of the performance tendency. I want to evaluate the influence that one specific variable has on the project performance, so changes made for each simulation run affect only one particular variable is only applied to one particular variable. Each model is not simulated by a combination of multiple variable changes. My goal is to find factors that

management can take advantage of by adjusting its value to reduce unnecessary delay, utilize slack time and complete project early. In the following section, I will demonstration my findings for each variable.

DLINCT Analysis – Average Delay in Incorporating Discovered Tasks

For each simulation, I plot the 'planned slack time remaining' (as PSTRM in the equations) to show the size of 'surprises' that the organization could potentially handle without overtime at any point in the project. This value also shows the importance of finding as many issues as possible and as soon as possible. Some of the discovered tasks can be absorbed by the team in the early phase of the project, but at certain point of time when there is a big amount of undiscovered job tasks, management decides to re-evaluate the project and estimate time and effort needed to complete the project. And further adjustment of man-days is triggered by the possibility of behind the schedule instead of the discovery of additional tasks. Figure *4* helps us visualize the differences of Daily Slack Time and other factors in Abdel-Hamid's model and the modified model with Average Delay in Incorporating Discovered Tasks (as DLINCT in the equations) set at 5 unit. From the graphs, we can see that Total Workforce (as TOTWF in the equations) peaks at a lower level; less slack time (as DST in the equations) is being used during the second half of the project; a greater amount of slack has been preserved around day 300.



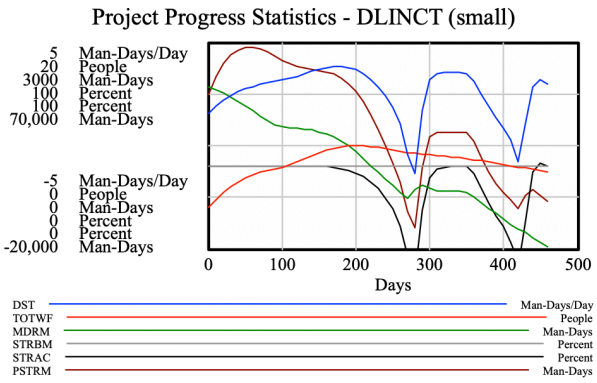


Figure 4: *Comparison of Daily Slack Time on Project Progress – Original VS DLINCT (small)*

The percentage of 'undiscovered tasks discovered per day' is a variable that increases in value as the project progresses. As the additional discovered tasks increase, they also need to be incorporated into the project plan. Incorporation early simply means that perception of project status is adjusted to be closer to actual status earlier – incorporating this information into plans usually implies employees also obtain better information about the project earlier. With earlier awareness of project status, employees can adjust their slack time wiser. With shorter delay in incorporating tasks, employees have a better understanding of the whole project status, which helps them adjust to a more appropriate work pace earlier. This change requires organization to be more integrated and efficient for gathering information, analyzing, reporting and making decisions.

Figure *5* shows the comparison of metrics when different values of Average Delay in Incorporating Discovered Tasks (as DLINCT in the equations) are applied to the model. By analyzing and comparing the simulation plots, we can see that DLINCT has significant influence on slack time. Shorter delays lead to increased work intensity (less invested in slack time) earlier, and to more realistic planning. As a result, slightly more work gets done earlier, and the organization is better able to deal with variations in remaining work during the later stage of the project. This improved behavior with respect to work intensity and greater retained flexibility allows the organization to complete the project with lower total work force. The dip in the second 'overwork period' is much less severe than for the two runs with larger delays.

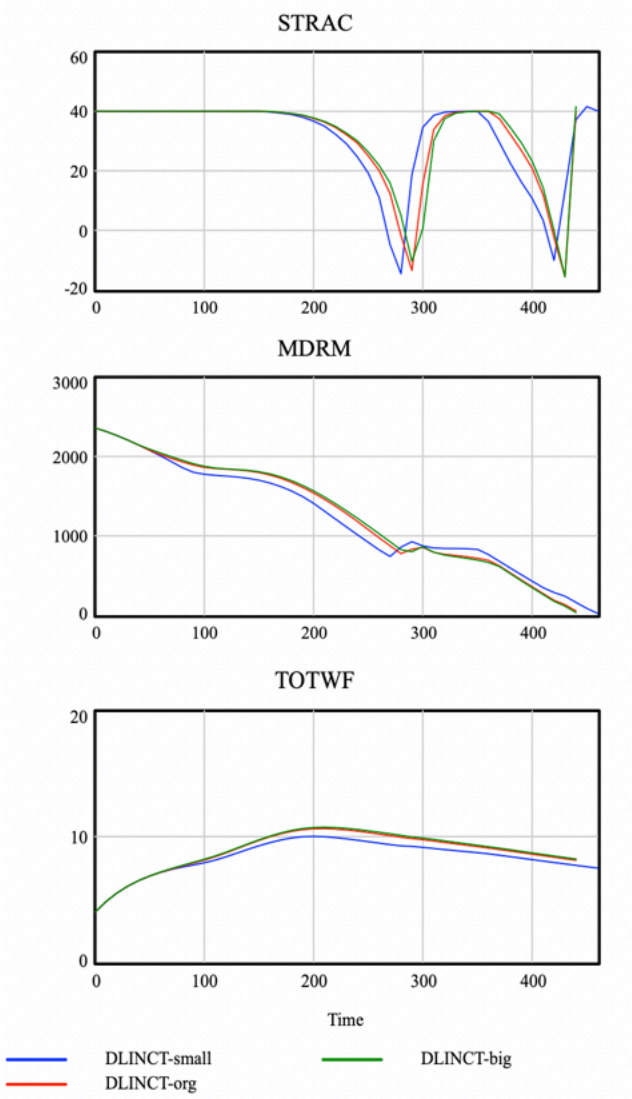


Figure 5: *Key Function Comparison with Different DLINCT Values*

AQADLY Analysis – Average Delay for Quality Assurance

The objective of quality assurance (QA) is to detect errors, generally before the project moves into system integration. Because it is impossible to tell whether all the errors have been detected, it is difficult to decide whether enough effort has been invested in QA. Furthermore, there is dis-incentives for working harder at QA. Under this circumstance, it becomes easy to rationalize the QA job and cut corners on QA when the project becomes seriously behind.

Figure *6* represents the differences of applying different AQADLY values in simulations.

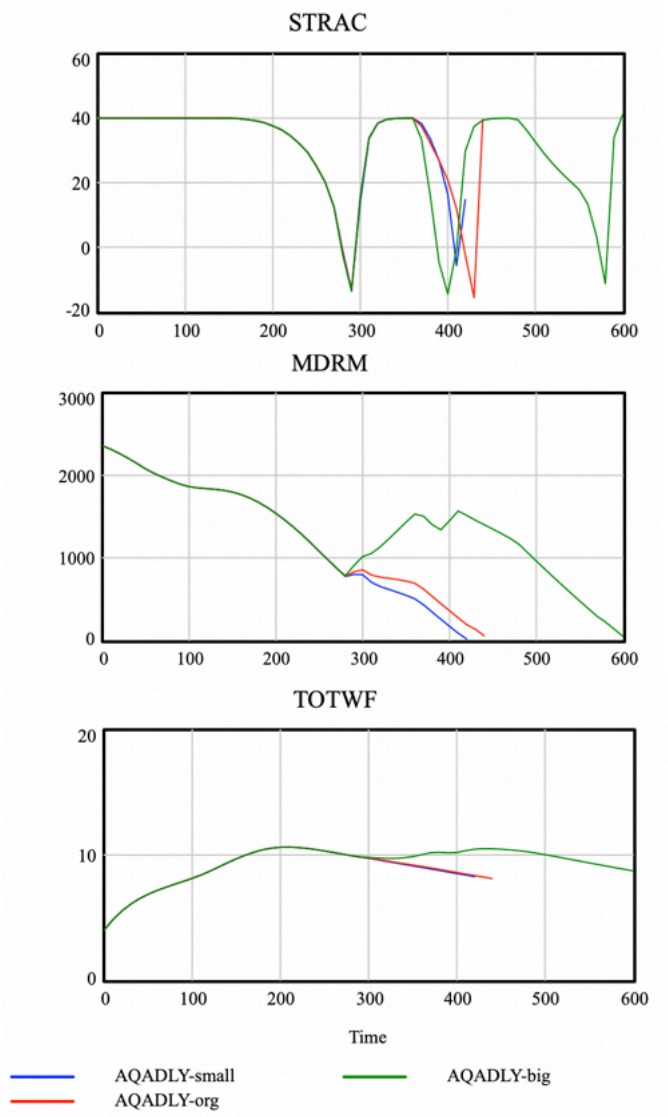
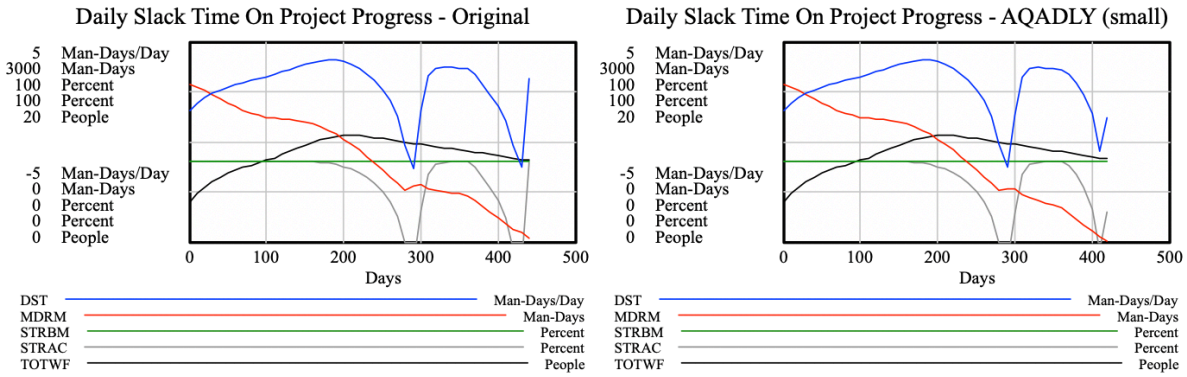


Figure 6: *Key Function Comparison with Different AQADLY Values*

QA effort is planned and allocated in a fixed schedule of group activities. During the QA window, all tasks developed since the last one was are supposed to be processed. Even when QA activities are suspended or relaxed, there are no backlogs developed in the QA pipeline. Backlogs develop only when QA section is suspended for a while, the requirement for QA is also suspended, not postponed [11][12]. Shortening the delay in QA (for example scheduling a meeting faster) does definitely improve the QA latency. With accurate

information on coding quality, employees tend to allocate their slack time wisely. In Abdel-Hamid's model the average QA delay is set to two weeks. With a shortened QA delay, the Man-Days Remaining (as MDRM in the equations) slope is quite sharp after the second slack time peak, and the work remaining decreases quicker. The interval of slack time is smaller as well. At day 420, there are only around 13 man-days left to finish the work, compared to 188 man-days left in the original plan. One possible reason is that the employees tend to respond quickly to the problems found in the program evaluation process, identify the problem, and fix it in a timely manner. At the current work pace, the project may pass the due date, thus, employees start to work overtime on day 280 in order to catch up with the schedule. The overwork lasts around two weeks to catch up with the scheduled plan for both original and Small AQADLY mode. At day 405, employees start worrying about the schedule and tend to work overtime in the AQADLY mode. Figure *7* represents the project progress in terms of daily slack time, man-days remaining and total workforce for each simulation.

Figure 7: *Comparison of Daily Slack Time – Original VS AQADLY (small)*



Similar competitive runs are examined on variable Desired Rework Delay (as DESRWD in the equations), Reporting Delay (as RPTDLY in the equations) individually. We also evaluated variable Exhaustion Depletion Delay Time (as EXHDDY in the equations). Although changing EXHDDY clearly impacts delivery time, it represents intrinsic

limitations of human beings [8]. The modification of EXHDDY does not really represent 'controls' that management team can adjust to improve the project outcome. Over all the parameters have been examined in Abdel-Hamid's model, we can observe that Average Delay in Incorporating Discovered Tasks (as DLINCT in the equations) is correlated to the Daily Slack Time at a certain degree. Average Delay for QA (as AQADLY in the equations) does have influence on daily slack time, and with the decrease of its value, the project can be completed a little earlier. This reduction does not require allocation of more manpower in quality assurance, but the code quality can be improved at a relatively lower cost. A shorter average QA delay provides the employees more accurate information about the coding quality, so they can allocate slack time more wisely, work on quality assurance earlier, detect errors sooner. Errors which are potentially generated by coding mistakes can be detected in a timely manner. Reducing delay in performing QA leads to earlier adjustments in work-intensity, with various secondary effects.

FREFTS Analysis – Fraction of Effort for System Testing

In the most rigid waterfall software development projects, each phase must be completed before the next phase begins and no overlapping is allowed in the process. All these phases are cascaded one after another. Integration and testing need to wait until system design and implementation are completed. The weakness of the traditional approach is that errors generated during the design or development process, will be carried through the development process and not discovered until the testing phase. Alberts estimates that errors from design phase are 2.5 times more costly to detect and correct [12]. These cumulative errors are a heavy burden for the project, especially if these errors are from the design phase. The traditional software development methodologies do not schedule employees working on system testing until code completion, but modern technologies allow employees to start

integration testing as soon as the project starts. Modern technologies like agile development are characterized by short bursts of development planning, developing/testing, evaluating with substantial co-testing of product, such that it is proposed that a separate testing stream is not necessary. System testing can happen in conjunction with programming. It will be valuable if we can evaluate the effect of system testing timing on the development performance of the project.

Integration testing is called ‘system testing’ in Abdel-Hamid’s model. The objective of integration testing is to test all tasks that have been developed to detect and correct any remaining (active and/or passive) errors [8]. After manpower is allocated to training, quality assurance, and rework activities, the remaining manpower is used for the development of software products. Abdel-Hamid stated that the allocation continues until management perceives that most of software development tasks are completed at which point the integration testing begins and manpower is allocated to testing. The switch in manpower allocation is governed by the variable Fraction of Effort for System Testing (as FREFTS, a tabular function or lookup table in the equations).

Abdel-Hamid assumes in his model that employees begin integration testing after 80% of tasks are perceived to be complete. Thus, FREFTS is the key parameter controlling the timing of integration testing. As a significant subsystem of the whole software development process and an important indicator of project status, we want to know if the timing of integration testing has significant influence on the project completion date and work intensity.

In modern approaches, employees do not delay the testing activities until the system design and implementation is finished in the linear-sequential life cycle. They can start

integration testing as early as when only 5% of perceived tasks completed. Since the system integration testing is conducted quite early, we expect that employees receive immediate feedback on the project status and detect the perceived shortage in man-days in a timely manner. With more accurate information about the project, employees can boost the hours they allocate to the project or get prepared for assigning proper manpower to what they perceive is necessary to handle all the perceived shortage in man-days.

FREFTS specifies, for each point in the project, how manpower is apportioned between development and testing. The original FREFTS function used in Abdel-Hamid's model begins allocating manpower to testing after 80% of the tasks are complete. The function then begins shifting manpower from development to testing relatively quickly, reaching a point where 100% of manpower is devoted to testing at about the same time as the project completes. (See Figure *8* (a)). At x axis point 0.8, the value of FREFTS is zero, since none of the manpower is allocated to system integration testing. All the manpower effort will be allocated to the system testing when 100% of the perceived development tasks has been completed.

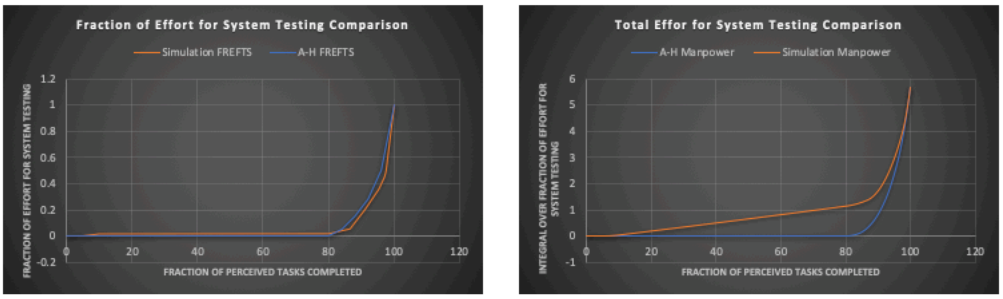
I have run four system dynamic simulations in my research and did the comparison with the original simulation in Abdel-Hamid's model. one is to start integration testing at the time point of 5% perceived tasks completed as the Extremely Early model, one is to start integration testing at the time point of 40% perceived tasks completed as the Modest Early model, one is to start integration testing at the time point of 60% perceived tasks completed as the Early model, one is to start integration testing at the time point of 94% perceived tasks completed as the Extremely Late model, compared to the one of 80% perceived tasks completed as the Late model which is implemented in Abdel-Hamid's model.

Since we only want to evaluate the influence of starting time on system integration testing, we need keep the aggregate portion of manpower applied to testing relatively unchanged. To conduct the comparison without any bias, the same amount of manpower is assigned to QA in all simulations. This parity in resource is achieved by keeping the area under the FREFTS curve the same as the area in the original model simulation. The modification of the starting time of system integration testing can be realized by changing the profile of the plot in the graph.

I constructed new lookup tables for fraction of effort on system testing (FREFTS) variations: Late, Extremely Early, etc. As Figure *8* shows, I constructed new lookup tables for presenting the FREFTS profile for Late model and Extremely Early model, and their corresponding integrals. My intension was to have the aggregate fraction of aggregate manpower applied to the project be the same in both experiments. That means the integral across this function needs to be the same. If the area under the function A-H FREFTS and Simulation FREFTS is the same, it indicates the allocation of manpower for both models is similar. This test increases our confidence that we are evaluating the independent effect of testing timing. The only difference is the timing of starting system testing which is the time point at x-axis where its corresponding value on y-axis is zero in Figure *8*.

FREFTS is nothing but a function. In Figure *8*(a), the x-axis represents the percentage of perceived tasks completed and the y-axis represents the percentage of effort allocated to integration testing. I want to examine if the starting time of integration testing has influence on the project performance with the comparable total manpower applied to each model. The integral of both FREFTS functions is the same, which also means the total manpower used in each model should be similar. A-H Manpower and Simulation Manpower in Figure *8*(b) are

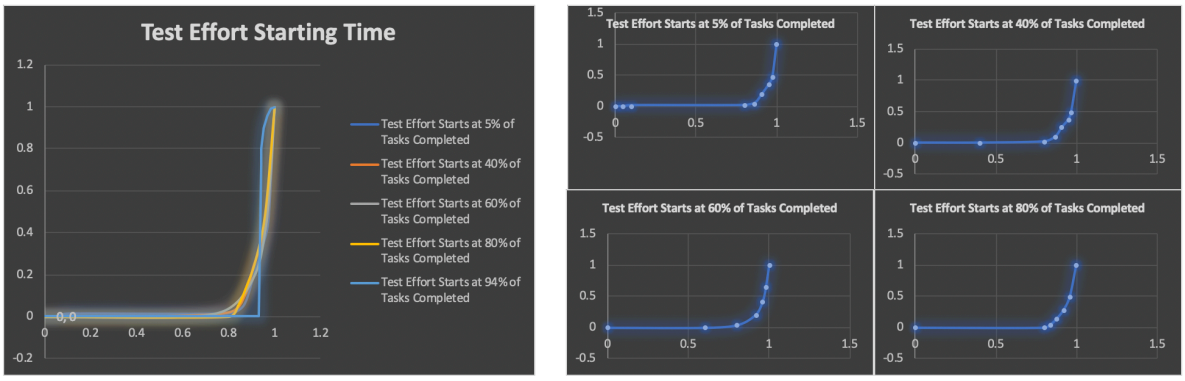
the accumulative value for A-H FREFTS and Simulation FREFTS, respectively. Both lines start at origin when 0% of perceived tasks has been completed, the cumulative effort is 0; and when 100% of perceived tasks has been completed, the cumulative effort has been allocated is the same. The project schedule is initially planned as 360 man-days in Abdel-Hamid's model. With 5% of tasks being completed, the employees have worked on the project for 18 days – a little bit more than one usual sprint, when they start working on system testing.



(a) (b)

Figure 8: *FREFTS Lookup Table Plot & Integral*

Figure *9* shows the results of computing a numerical approximation to the integrals of FREFTS when the integration testing starts until the project reaches about 5% completion, 40% completion, 60% completion, 80% completion and 94% completion. Figure *9*(a) compares all the FREFTS plots in one graph and Figure *9*(b) shows each individual FREFTS plot with different integration testing starting time.



(a) (b)

Figure 9: *Comparison of FREFTS Lookup Table*

However, Figure *10* indicates that along with reducing the total overtime, simple maintaining an equivalence in the integral of the FREFTS function, however, proved inadequate to maintain in terms of total man days spent in integration testing. From the comparison graph we can see that modest early model is our best performance model in terms of reducing slack time, but it actually demands more total manpower. I have not been able to isolate the causal mechanism for this variation. Total tasks developed, total errors generated, and other major cost drivers remain very similar across the simulation runs, but both test time and total project time behave non-linearly.

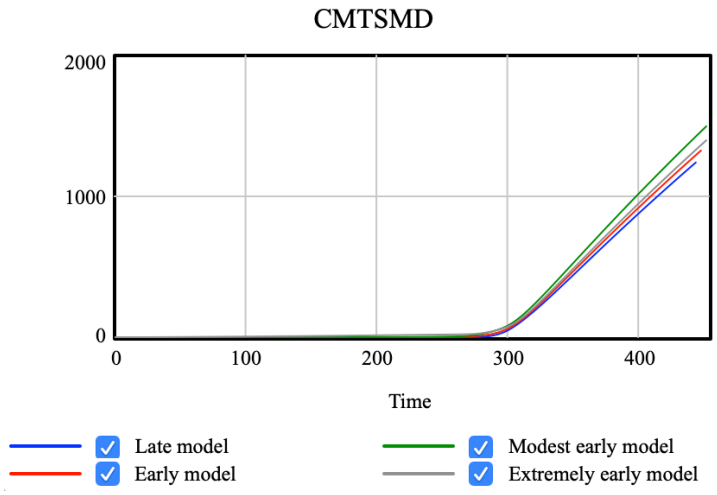


Figure 10: *Overall Cumulative Testing in Man-Days*

Next, I am going to evaluate each individual variable I have identified that have potential influence on project performance. Perceived Shortage in Man Days (PMDSHR in the model equations) represents the difference between the total effort needed to complete the project and the total effort that is actual remaining. Maximum Shortage in Man-Days to be Handled (MAXSHR in the model equations) constitutes a threshold determining whether the shortage can be handled. Total Job Size in Man-Days (JBSZMD in the model equations) represents the total needed man-days to complete the project. Actual Slack Time (STRAC in

the model equations) represents the actual fraction of a man-day on project that defines the fraction of daily hours allocated to project-related work by employees. Cumulative Errors Generated Directly During Working (CUMERG in the model equations) represents the error generated during the software development process. We choose to examine five different variables for each model, since these five variables reflect the information being used to determine the project completion data, work intensity and the errors generated during the process of software development.

Variable – PMDSHR Analysis

Perceived shortage in man-days remaining (as PMDSHR in the equations) is a key driver for the work intensity and project schedule changes. It represents information that may influence people's behavior such that employees work harder, management hire more people, or adjusts the completion date. Discovering the whole truth about the project status early gives the management and the team more time to response. If the project progress was found out at the very late stages of the development, there would have no time or no enough time to make any adjustments.

Notice that even with early testing, the team still encounters the shortage problem, but this can be detected in the early stage of the project. The early discovery in man-days shortage provides the team or the management enough time to make arrangements. Figure *11*(a) represents the Perceived Shortage in Man-Days for each simulation with different integration testing starting time. We expect that with early system integration testing, the team can recognize perceived shortage earlier, and after the team makes adjustments based on the shortage estimation, they encounter less shortage in the next shortage peak. Addressing these shortages earlier smooths out the workload in the later portion of the

project. The simulation of Extremely Early model perceived tasks completed is a little off the expectation, we will discuss it later.

The first peak in perceived man-day shortage in the Modest Early model simulation is at the 244th day and the first peak in perceived man-day shortage in the original Abdel-Hamid's model is at the 280th day showed in Figure *11*(b). Based on the first overshoot in man-day shortage, starting testing early gives the team 36 days to make response. We can see that the first peak in the Modest Early model simulation overshoots a little bit more compared to the first peak in the Late model, the overestimation gives the team plenty leverage to adjust the project. It leads to the second wave much smoother than the second wave in the Late model. That means after the first adjustment to the estimation of man days in shortage, the team can expect less shortage for the second shortage peak. That is beneficial to the project and the team. The shortage in man days is 80 at the second shortage peak day 358th in the Modest Early model simulation and it smoothed out to the end of the project completion, compared to 125 at the 402th day, the same stage in the Late model simulation. With previous expectation, we run simulations for the model and the simulations show that the earlier we start system integration testing, the earlier we can notice the shortage in man-days, the earlier the team can make adjustments. With early feedback from the perceived man-days in shortage, the project can be adjusted either by hiring more people, pushing employees work harder or compressing the slack time.

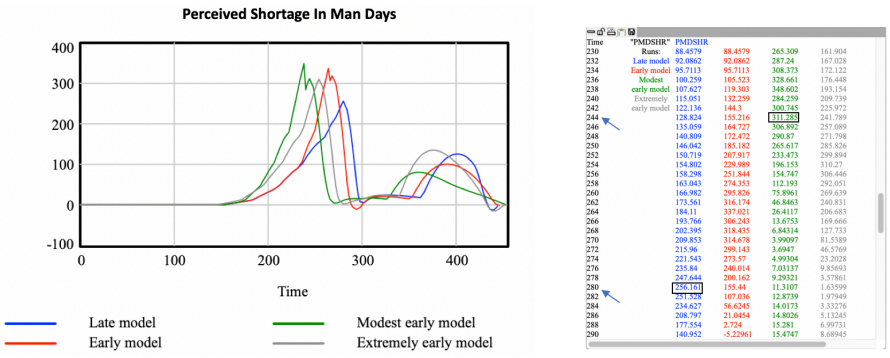


Figure 11: *Comparison Plot of Perceived Shortage in Man-Days & Table Values*

Variable – MAXSHR Analysis

Most projects are likely to be underestimated, since there is a powerful tendency to focus on highly visible mainline components of the software and miss or overlook the unobtrusive components [12]. When the project is behind schedule, the organization will tend to increase work intensity in an effort to catch up. Abdel-Hamid proposed that two factors determine the level to which the 'actual fraction of man-day on project' is boosted. The first factor is Perceived Shortage in Man-days (as PMDSHR discussed above). The second factor is the 'Maximum Shortage in Man-Days that can be Handled' (as MAXSHR in the equations). If the perceived shortage is greater than the maximum shortage the employees are willing to handle, Abdel-Hamid proposes that employees could be motivated to work harder to accommodate the maximum value, while arranging with management to extend the schedule to handle what exceeds the 'maximum shortage in man-days that can be handled' [7]. But the maximum shortage is not a constant parameter. As employees work harder to deal with shortages in man-days, their tolerance for working harder decreases and the maximum shortage value decreases as well. Based on the perceived shortage in man-days outcome in our simulations, since the Modest Early model simulation has the earliest and highest value around day 240th, we expect the maximum shortage to be handled in this

simulation starts dropping earlier and quicker; for the second perceived shortage in man-days' peak, we expect that MAXSHR has a larger value compared to others. According to Figure *12* the MAXSHR comparison plots generated by the simulations, the outcome exactly meets our expectations. Starting integration testing early decreases the team's tolerance to handle shortages, leads to the decrease of the value for the maximum shortage that can be handled. However, since the team has already handled the first shortage, their tolerance has been adjusted and the maximum shortage that can be handled for the second shortage will increase a little.

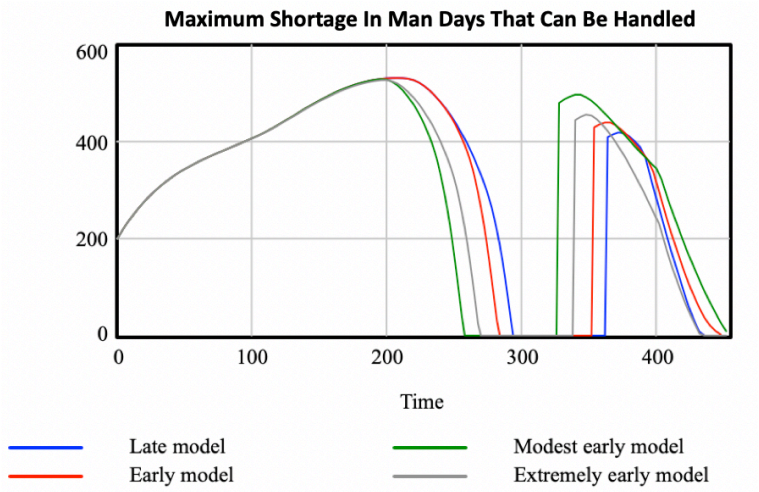


Figure 12: *Comparison Plot of Maximum Shortage in Man-Days that can be Handled*

Variable - JBSZMD Analysis

'The Dynamics of Research and Development' was the first scholarly effort to apply system dynamics methodology to software project management within a research and development environment. In the paper, Edwards B. Roberts demonstrated that two factors significantly affect the initial estimate of the project job size: one is the organization's previous experience, the other is the general over-all tendency to underestimate the job size.

Total job size in man-days can be adjusted upward or downward to what is perceived as its new value.

Management first perceives the shortage in man-days around day 148 in the Modest Early mode compared to day 156 in the original model. The total job size is smaller than the value in the original model until day 254 where the total job size starts rising. Abdel-Hamid observes that the adjustment process is not an instantaneous one, instead it takes place over a time defined as the 'Delay in Adjusting the Job's Size in Man-Days'. In practice, the delay in adjusting the job size is from two days to five working days, Abdel-Hamid used three days in the model. When the revised value of job size in man-days equals the total job size in man-days, there is no adjustment needed. But if the revised value of job size increases, in response to the rise, the value of total job size in Man-days rises rapidly to achieve the goal. Figure *13* shows that if the system testing starts early, the revised value of job size is estimated greater, which leads to a steep rise in total job size. Note that the flattening part comes early in the Modest Early model. That means the management or the team experience stable projections during the latter part of the project, there is no last-minute surprise need in manpower.

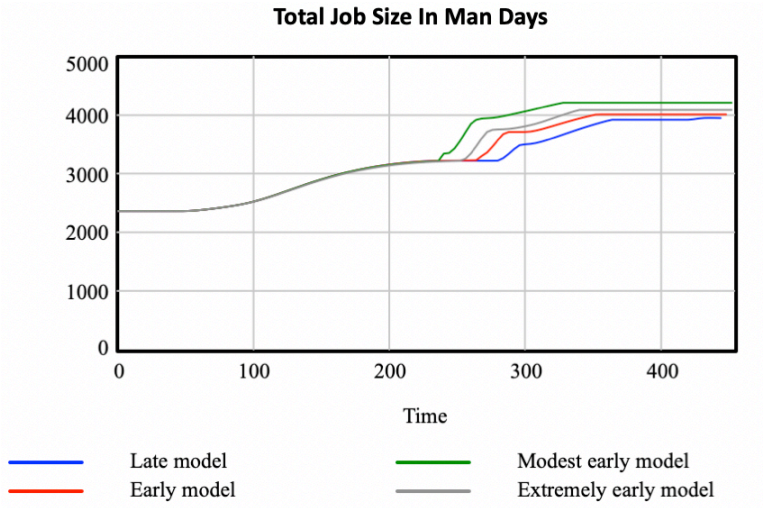


Figure 13: *Comparison Plot of Total Job Size in Man-Days*

Variable – STRAC Analysis

Abdel-Hamid points out that based on certain research conducted in the software development industry, most of the estimates of slack time are clustered within 50% - 70% range [6]. The ideal slack time rate should be relatively constant between the industrial average 40% and 20%. Consequently, Abdel-Hamid set the value of nominal fraction of a man-day on project to 60%, which means that the loss in productivity – the daily slack time rate remains constant at 40% level over the project. We expect that if the system integration testing starts earlier, the slack time rate can become relatively stable and remain between 40% and 20%. As Figure *14* shows: Modest Early model simulation starting system testing at 40% of perceived tasks completed provides the most stable slack time rate, especially at the latter stage of the project development process. Meantime, starting testing early allows the organization to begin utilizing slack time earlier and adjust their work pace to meet the deadline. From the graph we can see that while approaching the latter stage of the project, the slack time rate remains in the range of 40% - 20% at a relatively constant level. Constant slack time remained represents the tasks in the project has been handled properly, there is no steep overshoot or big shortage in manpower. With timely feedback about the project status and less information delay, employees are able to respond to the backlog properly. We want as many time points stay in 40%-20% slack time range as possible during the project time interval. I counted the total amount of days slack time points in the specific time range (Table 2), it turns out that the Modest Early model has most ideal points falling in the range. Overall, the earlier the system integration testing starts, the more stable the slack time is during the development process.

Table 2: *Time Points Count Comparison for Actual Slack Time Rate in range 40% - 20%*

Time Points Count Comparison for Actual Slack Time Rate in 40%-20%				
Model	Late Model	Extremely Early Model	Early Model	Modest Early Model
Count	182	173	189	204

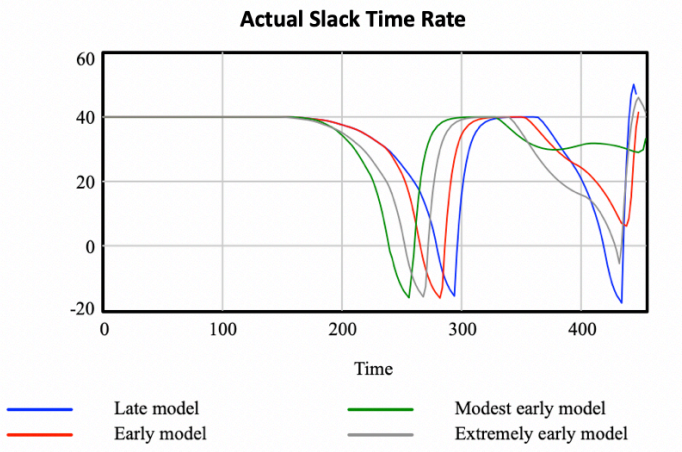


Figure 14: *Comparison Plot of Actual Slack Time Rate*

Variable - CUMERG Analysis

Abdel-Hamid assumes that all errors which are detected will be reworked before the project terminates, the errors detected are always fewer than the number of errors which are generated (i.e., which exist in the project). In practice some of the errors remain in the product even after completing system testing. It would seem reasonable to assume that if system testing starts earlier, fewer cumulative reworked errors would be generated and carried on.

Cumulative Errors Generated Directly During Work (CUMERG in the equations) is the multiplication of software development rate and errors generated per task. Errors are generated at a constant rate based on code size. And errors are regenerated at a certain rate based on the size of the code base and the age of the existing errors. The change in the

numbers of errors generated is probably related more to the extension of the project life or to the extension of the development phase than anything else. Figure *15* represents the cumulative errors generated for those four simulations. In our simulations, the timing of the generation was a little different, but all four simulation lines wind up at almost exactly the same place.

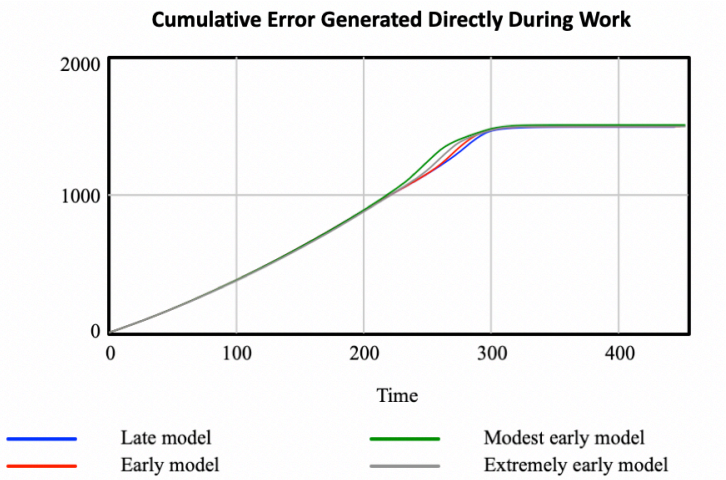


Figure 15: *Comparison Plot of Cumulative Errors Generated Directly During Working*

Extremely Early Model Analysis

In common sense, based on our analysis for each variable above, early integration testing is able to provide timely feedback on the project status. With the accurate information about the project progress, the development team should be able to adjust their behavior to accommodate to the project in a timely manner. That being said, the simulation with integration testing starting at 5% completion should perform the best. However, according to the individual variable analysis and comparison graphs we got above: it shows that Extremely Early model is not the best performer. My perspective on this consequence is that in Extremely Early model simulation, there is only tiny piece of corresponding manpower can be allocated to system integration testing, not enough effort can be applied to make any differences. For example, when there is only 5% of perceived tasks completed, even with

100% manpower allocated to integration testing, there is only 86 man-days (assume 4 employees in the team: 430*5%*4) used in the testing. Compared to the Modest Early model, it has 688 man-days allocated in the testing.

With all the questions for the Extremely Early model, I run several simulations: 1. Starting testing when 5% of tasks completion; 2. Starting testing when 5% of tasks completion with 20% more of manpower allocated to the current stage point; 3. Starting testing when 5% of tasks completion with 20% more of manpower allocated to the current stage point but only 60% of overall manpower applied to the project. All these three models are compared to the best previous performing model – the Modest Early model. Figure *16* proves our suspicion about the project performance when starts integration testing at 5% tasks completion. We can see that very early integration testing cannot guarantee the best performance. With all the simulations we run, the Modest Early model performs the best.

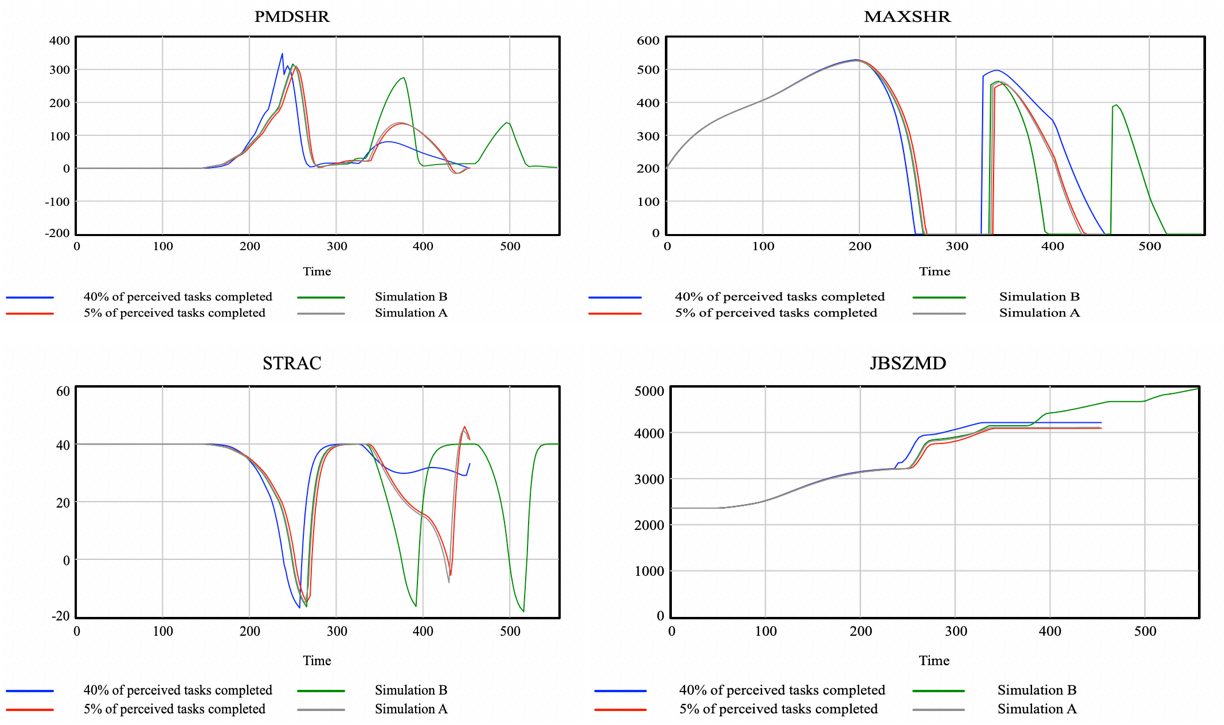


Figure 16: *Variable Performance in Extremely Early Model*

Overall, according to the observation and analysis on our system dynamic models, if employees start system testing at the early stage of the software project development phrase, they can receive more accurate information about the project status. With all the information of the project, employees can boost the hours allocated to the project. As employees allocate their slack time wisely, some aspects of the project get improved like slack time handling. the overall performance of the software development gets improved. In general, simulations at early integration testing starting time perform better than the ones at late integration testing starting time, for example the Modest Early model simulation surpasses the Early model simulation which surpasses the Late model simulation. Notice that the earliest does not necessarily mean the best. From the comparison of the overall performance, we can see that starting system integration testing at 40% of perceived tasks completion performs much better than starting system integration testing at 5% of perceived tasks completion with regard to daily slack time, perceived shortage in man-days be handled.

Final Model & Evaluation

Based on the analysis above, I built the final model with the best value for each variable analyzed before. The combined model used DLINCT as 5, AQADLY as 5 with system integration testing starting at 40% of perceived tasks completed. We expect that the final model will perform much better than the original Abdel-Hamid's model in terms of handling information delay, dealing with slack time, completing the project, etc. Figure *17* shows the performance comparison of the final model and the original Abdel-Hamid model. The actual slack time rate, as we expected, remains at a relatively stable level at the latter stage of the project in the final model. Figure *19* shows the comparison of cumulative errors generated for each simulation. It shows that our final model has the fewest errors generated directly during the software development process.

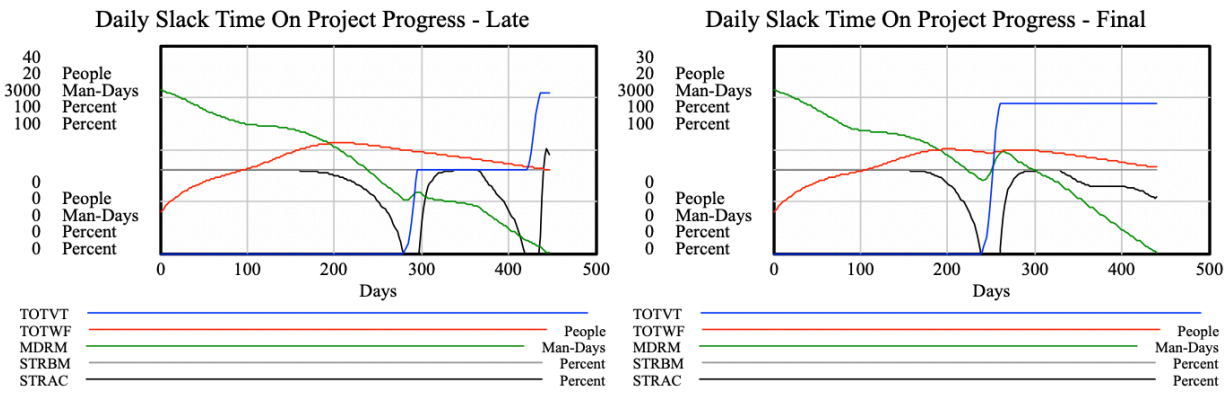


Figure 17: *Comparison of Daily Slack Time on Project Progress – Original Model vs Final Model*

Meantime, we expect our final model has smaller total overtime spent on the project. Figure *18* represents the aggregated total project overtime. From the values we can tell that the total overtime allocated on Final model is much smaller than that in the original Late model: 21.7239 vs 30.9391. The simulations have proved that delays in accurate information correlate with inappropriate decisions about work-intensity. Some of these delays seem to have an independent effect, some (like shifting testing starting time) have a complex effect that influences much more than just slack time and work-intensity.

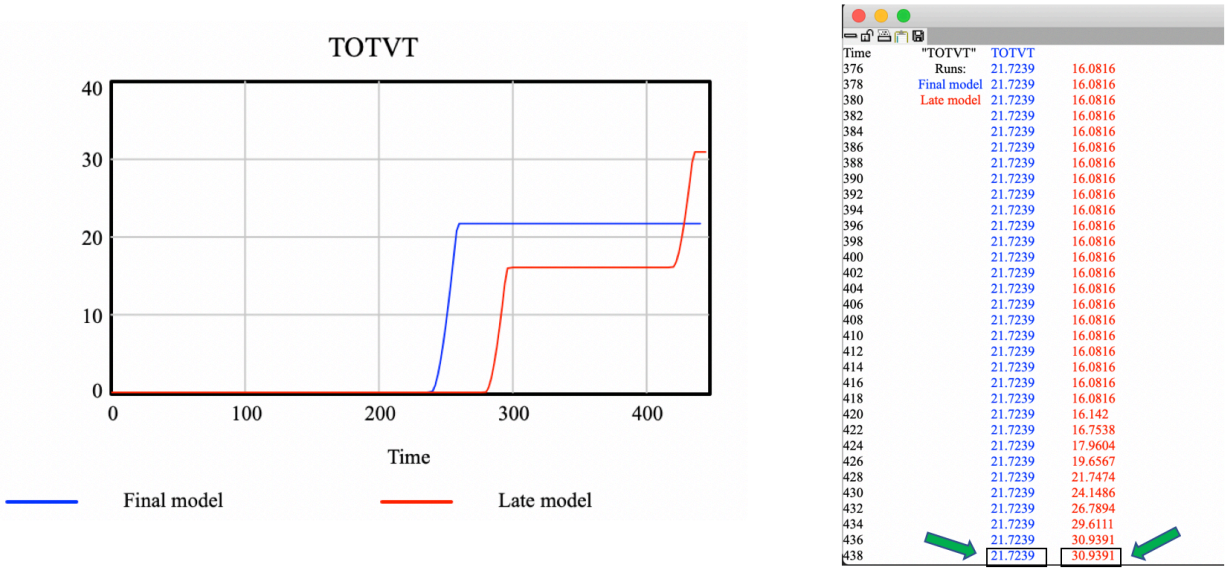


Figure 18: *Aggregated Project Overtime – Original Model vs Final Model*

Comparison of Cumulative Errors Generated Directly During Working					
Model	Late Model	Extremely Early Model	Early Model	Modest Early Model	Final Model
Count	1495	1498	1499	1509	1493

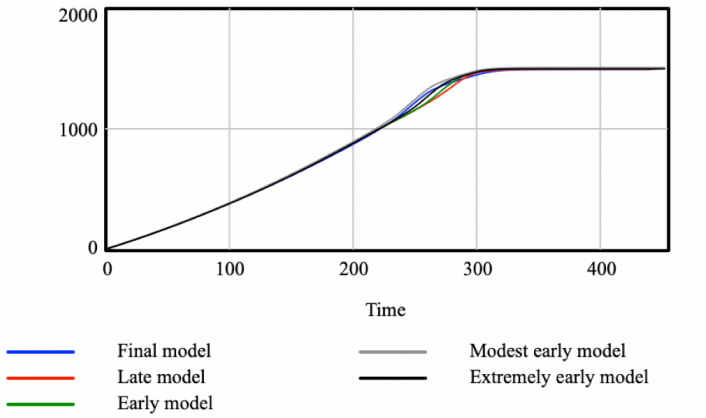


Figure 19: *Comparison of Cumulative Errors Generated Directly During Working*

CHAPTER 3. CONCLUSION, LIMITATIONS AND FUTURE WORK

Conclusion

Abdel-Hamid's model is based on the traditional software development methodologies such as the waterfall method, but our findings are applicable to modern methodologies as well, such as Scrum. In particular, Scrum includes practices that are meant to reduce information delay which we found had an impact on software development. For example, pair programming folds a key component of quality assurance (namely code review and design reviews) into the code writing process so that there is almost zero delay between writing code and reviewing code. Scrum, like many other agile practices, uses continuous integration, so there are seldom more than a few hours delay between system integration testing results and employees' awareness of those results. Finally, testing begins immediately and is as near in scope to system integration testing as possible. Thus, one interpretation of these results is that our experience and understanding of waterfall development already contains the needs for understanding how certain aspects of agile development might work.

Limitations

First, we have modified and selected particular values for each parameter examined in the simulations in order to attain the best performance model. However, those values applied in our designated model may not be applicable in the industry. For example, we assigned five days to the average delay in QA, which means it takes five days from the coding to quality assurance testing. In agile practice, it is only half of the sprint in most Scrum. For Scrum teams (since many use two-week sprints), this delay should be quite easily achieved. Second, our focus is to improve the project performance through investigating the relationships of slack time and integration testing timing. The original Abdel-Hamid's model has a big scope,

but in our simulations, we mainly focus on a small part of his model to examine the influence of delays. Third, since our goal is to improve the project performance by eliminating unnecessary delays in the development process, our choices for simulation parameters are the ones that delay related. Some importance stages may have not been covered in our research. Overall, this research is limited to investigating project performance with regard to delays caused by slack time allocation, integration testing timing, manpower allocation, and project completion date.

Future Work

According to the outcome of the simulation runs, the total project size is non-linearly related to the changes in the variable of fraction of effort on system testing (as FREFTS in the equations). We wonder if there are other aspects of the model that interfere with effective early system integration testing. If there indeed exist some factors that have not brought to our attention, we have to ask ourselves whether there are different, realistic parameter choices that can capture the benefit with respect to slack time management, without incurring additional overall costs. Finally, all the simulations I have run are based on Abdel-Hamid's original model, but if a re-examination of other existing software development models will help us reveal other aspects related to slack time management but with exact same overall costs. I want to say that this research work shows that there are seeds buried in existing traditional science that can improve our understanding of non-traditional methods like Agile methodology - Scrum.

REFERENCES

[1] White, A. S. (2014). An agile project system dynamics simulation model. International Journal of Information Technologies and Systems Approach (IJITSA), 7(1), 55-79.

[2] Sterman, J. D. (2000) Business dynamics: System thinking and modeling for a complex world, McGraw-Hill.

[3] L. Cao, "Modeling dynamics in agile software development," Ph.D., Georgia State University, United States -- Georgia, 2005.

[4] Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. IEEE software, 25(1).

[5] Madachy, R. J. (2007). Software process dynamics. John Wiley & Sons.

[6] Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting agile development methodologies. European Journal of Information Systems, 18(4), 332-343.

[7] Abdel-Hamid, T. K., Sengupta, K. and Ronan, D. (1993) Software project control: An experimental investigation of judgment with fallible information., TSE, 19, 603-612.

[8] Abdel-Hamid, T. K., & Madnick, S. E. (1991). Software project dynamics: an integrated approach (Vol. 1). Englewood Cliffs, NJ: Prentice Hall.

[9] Richardson, G. P., & Pugh III, A. I. (1981). Introduction to system dynamics modeling with DYNAMO. Productivity Press Inc.

[10] DeMarco, T. (2002). Slack: Getting past burnout, busywork, and the myth of total efficiency. Crown Business.

[11] Mitchell, J. R. (1980). Observations on the Use of Seven Structured Programming Techniques. In Proceedings,/f£ f Computer Society s 4th International Computer Software and Applications Conference (pp. 229-235).

[12] Alberts, D. S. (1976, June). The economics of software quality assurance. In Proceedings of the June 7-10, 1976, national computer conference and exposition (pp. 433-442). ACM.